Annals Hist Comput (1989) 10:277–303

A Historical Overview of Computer Architecture

RICHARD E. SMITH

Computer architecture concentrates on the logical aspects of computer design as opposed to physical or electronic aspects. The underlying logical design of most modern computers is still based on that of the earliest electronic computers despite decades of progress in electronic circuitry. The innovations that have occurred in computer architecture have been driven by two different goals: higher performance and lower cost. Performance driven improvements have yielded computer systems with increasingly higher computation speeds and throughput. Cost driven improvements have yielded systems that are easier to use and applicable to a broader range of automatic control problems. Improvements in electronic circuitry have not led directly to architectural innovations; computers that pioneered new circuit technologies usually relied on older architectural concepts.

*Categories and Subject Descriptors: K.2. [**Computing Milieux**]: History of Computing—Hardware. B.2. [**Arithmetic and Logic Structures**]: General, C.1. [**Computer Systems Organization**]: Processor Architectures. C.5. [**Computer Systems Organization**]: Computer System Implementation.*

General Terms: Design, economics, performance, reliability.

Introduction

A computer's architecture refers in part to the organization of its electronic components just as a building's architecture refers in part to its floor plan. Since a computer's purpose is problem solving through computation, its architecture also refers to the methods and capabilities it provides for automatic computation and control. Many systems for automatic computation and control have been built over the years using mechanical and electrical components (Goldstine 1972). Although many such systems have worked successfully and served as forerunners to modern electronic systems, the nonelectronic systems are beyond the scope of this narrative.

As with buildings, a computer's architectural significance is not directly determined by its development cost, fame, or popularity. Countless electronic computers have appeared over the past 45 years. Many computers achieved fame in part from innovative architecture though others succeeded entirely on the merit of innovations developed elsewhere. This narrative will concentrate primarily on the computer systems that took important architectural concepts and made them work.

Computer architecture first appeared in the late 1940s as researchers started work on the first modern electronic computer systems. The work is subtly different from that of the electronic circuit designer. Although many of the early computer designers concentrated primarily on circuit design, some of them concentrated on the *logical* aspects of the computer design. The computer architect abstracts beyond the level of electrical and electronic circuitry, working instead with idealized computing elements with explicit, logical, and well-defined properties.

The abstract view used by the computer architect is a product of the complexity of building

Author's Address: FMC Corporation, Advanced Systems Center, Mail Stop T210, Minneapolis, MN 55455.

computers. An architectural view of the computer fits well into a hierarchical view of the computer development process, thus making the process easier to comprehend and manage. The architectural view also allows designers to contemplate the functional capabilities of the computer without having to consider the electronic details at the same time. Computer architecture is founded on the reliable operation of certain building block circuits that implement a fundamental set of logical operations. Computer components can then be designed by combining these components and analyzing the results through manual examination or by using formal techniques of mathematical logic.

Fundamentally, the architecture of a computer involves four components: the central processor, the primary memory, the instruction set, and the input/output structure. The *central processor* has the primary task of controlling the system and performing all computations. The *primary memory* stores the program the computer is currently executing and whatever data is immediately needed by that program. The *input/output structure* determines how information gathering or distributing devices (e.g. terminals, printers, magnetic recording devices) may be attached to the computer and controlled by the central processor. The *instruction set* is not a physical component like the other three. Rather, it comprises all of the computational or control operations that the central processor can be instructed to perform. Programs that work correctly on a particular computer must use its instruction.

The basics of computer operation can be understood by examining Figure 1. Solid lines show paths that data follows while passing through the computer. Data is read from the computer's *input* into its *high speed memory*. Part or all of this data, incidentally, may be formatted according to the rules of the computer's instruction set and thus be used as the *program* that controls the computer's operation. Results produced by a program are stored in memory and can be sent from there to the computer's *output*.

Instructions from the program are read from the memory into the *control*, which issues control signals to other portions of the computer in order to perform the instructed action. The dashed lines in the diagram show the paths of control signals that direct the computer's actions. Under normal conditions the computer's circuitry will operate only in response to control signals generated by instructions. When an instruction performs an arithmetic operation, additional data is read from the memory into the *arithmetic unit*, combined according to the desired operation, and then written back into memory. The arithmetic unit and the control unit comprise the computer's central processor.

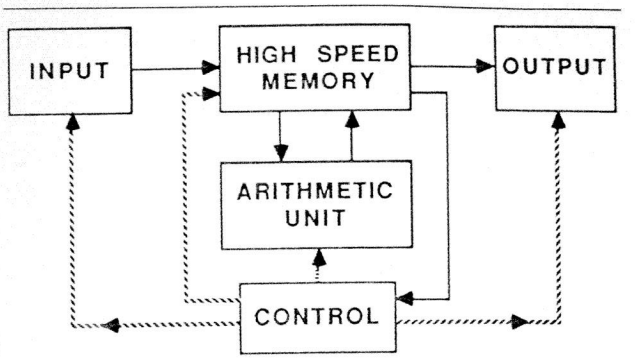

Figure 1. Block diagram of a digital computer after Smith (1948).

Innovation Through Adaptation

In a sense, much of computer architecture is unchanged since the late 1940s. The block diagram shown in Figure 1 is one of the earliest produced, yet it still gives an accurate portrayal of modern

Richard E. Smith is a Project Engineer on the staff of FMC Corporation's Advanced Systems Center in Minneapolis. His professional interests include robotics, multiprocessor systems, and artificial intelligence. He earned a B.S. degree from the Boston University College of Engineering in 1976, and an M.S. and Ph.D. from the University of Minnesota in 1983 and 1987, respectively. In 1987 he was an invited research fellow at the Centro Studi e Applicazioni in Tecnologie Avanzate in Bari, Italy. Prior to graduate school he served on the technical staff of Bolt, Beranek, and Newman, Inc. Dr. Smith's interest in computer architecture and its history is probably the result of having been raised among stacks of aging computer documentation and research reports.

computer operation. Architects then were still dealing with the same basic components—though in a less advanced form. There are many computers in use today that rely on few architectural concepts that appeared later than 1950; most computers rely on none introduced after 1965. Even recent concepts such as reduced instruction set computing (*RISC*) and *parallel processing* simply reintroduce older architectural ideas, adapting them to work in new systems. This is not meant to imply that innovation in computer architecture ended in the 1960s, but only to recognize the forerunners of recent topics in computer architecture.

This also illustrates another important point: the electronic technology being used by a computer designer is not necessarily an issue when looking at the computer architecture. When transistor circuits replaced vacuum tubes in the late 1950s, many computer manufacturers simply reissued transistorized versions of their vacuum tube architectures; they did nothing to exploit the architectural potentials of the new technology. Clearly, computer architecture can be independent of the underlying circuitry. The move from one "computer generation" to the next, in the popular sense of that term, simply reflected a change in the technology used to build the logic circuits; a new generation did not in itself herald a major advance in computuer architecture.

Often, though, the new technologies did lead to architectural innovation. The trend in computer circuitry has been towards circuits that are smaller, cheaper, and easier to fabricate. This trend has been exploited two ways: some architects use more circuitry to improve performance, while others build economical computing power by using more austere designs. Each of these paths has spawned its own innovations. Designers working for performance have found new ways of organizing the basic components of the computer to incorporate extra circuitry. Designers attempting to minimize cost have found new ways to provide inexpensive computing power—occasionally producing new architectural building blocks (e.g. the *microprocessor*) in the process.

This paper presents the history of computer architecture in four sections. The first section discusses the origins of computer architecture and the early days of electronic computer design. The second and third sections discuss two separate tracks in computing history that correspond to two different design goals. The second section follows the chronological development of computers designed primarily for performance. These systems are often called *mainframe* systems; the fastest are called *supercomputers*. The third section presents the development of austere computers designed for lower cost. These systems, when recognizable as individual computer systems, are the *minicomputers* and *microcomputers* common today. A final section presents a perspective on the history of computer architecture.

Origins of Computer Architecture

The first significant work on modern computer architecture resulted from the joint efforts of John von Neumann and the developers of the Electronic Numerical Integrator and Calculator, or ENIAC, at the University of Pennsylvania's Moore School of Electrical Engineering (Goldstine and Goldstine 1946). The ENIAC was the first important general purpose electronic calculating machine, and was developed by J. Presper Eckert and John W. Mauchly. The ENIAC was considered a *calculator* rather than a computer because of its inflexible approach to computation. The ENIAC consisted of several accumulating adders and several special function units such as multipliers. Calculations were set up by setting switches manually and plugging cables to interconnect the appropriate units, a time consuming task.

As the ENIAC approached completion, Eckert and Mauchly started planning a more powerful machine called the EDVAC, or Electronic Discrete Variable Automatic Computer. A major goal of the EDVAC design was to avoid ENIAC's cumbersome method of problem set-up by giving the EDVAC a memory that would store the instructions directing it to execute the desired calculation. By this means the computer could be devoted to a new task simply by placing different instructions in the memory. This became known as the *stored program concept.*

At the time these ideas were developed, John von Neumann, a distinguished mathematician at the Institute for Advanced Study (IAS) at Princeton, became a consultant to the EDVAC project, Figure 2. He recognized the significance of the stored program concept and soon drafted a report describing the design, carefully explaining the stored program concept. This manuscript, known as the "First Draft of a Report on the EDVAC," was circulated privately in June of 1945 (von Neumann 1945). Although never formally published, the draft report was widely circulated and had a significant impact. The document strongly influenced M. V. Wilkes in his design of the EDSAC, a pioneering computer built at the University of

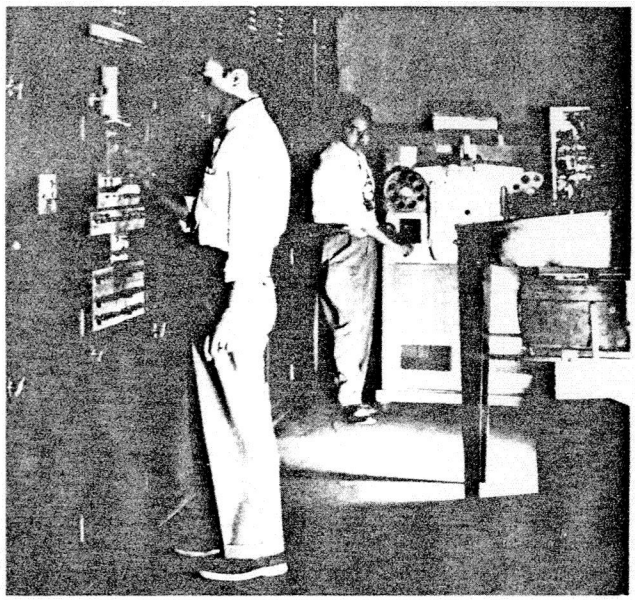

Figure 2. A general view of EDVAC, the electronic discrete variable computer, Aberdeen Proving Ground, Ballistic Research Laboratories.

Cambridge, England (Wilkes 1951b). It also had an important effect on the design of most early computers in the United States.

An important feature of the draft report was that it discussed the architecture of the EDVAC in terms of abstract computational elements instead of electronic circuitry. This was a different point of view from that taken by Eckert, the electronic designer of the ENIAC, who preferred to tackle computer design at the electronic level. Von Neumann considered abstraction an important part of computer design, and it played a major role in his discussions of computer architecture.

Back at the IAS in 1946 von Neumann collaborated with Arthur Burks and Herman Goldstine on the design of a new computer, eventually known as the IAS machine. In June of 1946 the designers at IAS produced a report entitled "Preliminary Discussions on the Logical Design of an Electronic Computing Instrument" (Burks, Goldstine, and von Neumann 1946). This report received even wider distribution than the EDVAC report and was the first officially distributed document describing the fundamentals of computer architecture. The IAS report also went into greater detail than the EDVAC report. The significance of the IAS report was so great that all subsequent stored program computers were referred to as *von Neumann type* machines.

Considering the wide distribution of the IAS report, it is not surprising that one can see the architecture of the IAS computer reflected in a broad range of subsequent machines. The Whirlwind (Everett 1951) at the Massachusetts Institute of Technology (MIT), the early computers at the University of Illinois (Robertson 1980), the JOHNNIAC at RAND Corporation (Gruenberger 1979), and the IBM 701/704/709 series of scientific computers at IBM (Hurd, 1981; Bashe, et al. 1986) were all heavily influenced by the IAS architecture. The three major attributes of the IAS design were the *stored program,* the use of the *binary number system,* and the *parallel arithmetic* architecture. The stored program concept has been discussed above. Binary arithmetic and parallel circuitry have also become very important architectural techniques. Most computer systems today also contain the three crucial design features of the IAS computer.

The interest in binary arithmetic was motivated by the importance of binary logic circuits. These circuits responded simply to the presence or absence of control signals and operated reliably at high speeds. Since numerical data was generally provided in decimal notation it had to be converted to a binary form for computation. The ENIAC handled this problem by building decimal computing circuits. The IAS report argued that number conversion between binary and decimal could be performed by the stored program with a net saving of memory and logic. At best, a decimal arithmetic circuit must use four binary circuits to handle one decimal digit representing values between 0 and 9. The same four binary circuits operating on binary data can handle numerical values between 0 and 15, yielding more capacity from the same amount of circuitry.

The choice of parallel arithmetic organization determines how numbers are processed and moved between units of the computer: the binary digits comprising a number may be transferred between units either serially or in parallel. In a serial organization, data is moved and processed a digit at a time. A serial arithmetic circuit, for example, would add two numbers a digit at a time. In a parallel organization, entire numbers are moved and processed by the circuitry. The choice is a trade between speed and circuit complexity. Serial arithmetic circuits are less complex while parallel circuits perform computations much faster. The IAS design opted for speed and chose a parallel organization.

Most discussions about computer design at the

time focussed on the implementation of key components, such as arithmetic or storage circuits, or their interconnections. A few techniques were found to reduce the enormous cost of computer design and implementation. Table 1 illustrates some of the tradeoffs that designers could make. Serial arithmetic and control units required fewer vacuum tubes than parallel units, making such machines more economical and easier to build. However, serial machines paid dearly in terms of computation speed, as shown by the amount of time it could take to add two numbers. Different types of high speed memory could provide greater or lesser amounts of memory, but increased sizes were often at the expense of computation speed.

A very important early development was that of *microprogramming*, developed by Maurice Wilkes while building the EDSAC computer (Wilkes 1951a; Wilkes and Stringer 1953). Wilkes observed that the problem of controlling the movement of data between components of a central processor was similar to the problem of performing a computation with a stored program computer. Microprogramming allowed complicated sequencing circuits to be implemented by simple and flexible diode matrices. In effect, the computer was programmed to control its own internal information flow, greatly simplifying computer design. This technique was also evident in the Whirlwind design (Everett and Swain 1947), though it was not called microprogramming and was apparently developed independently.

Another architectural feature developed in the late 1940s was the *index register*, which first appeared on the Manchester University Mark I (England) in early 1949 (Lavington 1975; Campbell-Kelly 1980). Index registers, called *B-lines* on the Mark I, greatly simplify the writing of programs that manipulate data in tabular or vector form. Each instruction in a program usually contains a numerical value that identifies the exact address in memory where it would find the data to use. This is inconvenient when handling vectors of numbers, since a program often needs to perform an identical instruction sequence on each number in the vector. The programmer would either have to give the computer an essentially identical instruction sequence for each element in the vector or else have the program modify its instruction sequence to refer to successive elements in the vector.

Index registers eliminated this problem. The index register contained a numerical value that could be added to the data address given in an instruction. When using an index register, the program's instructions would reference a different address in memory according to the value stored in the index register. Numbers in a vector were usually stored in successive storage locations, so an instruction sequence could be applied to successive numbers in a vector by incrementing the number stored in the index register.

It is important to remember that few, if any, designers of early computers had experience with a working computer. The pioneering computer architects had no preconceptions of what would work and what would not. By around 1950, the first computer projects were reaching completion. In June of 1948, an early prototype of the Manchester Mark I successfully ran a short program, probably the first in a stored program computer (Williams and Kilburn 1951; Lavington 1975). The first commercially produced computers, the American UNIVAC (Stern 1981) and the British Ferranti Mark 1 (Lavington 1975), were delivered in early 1951.

Early Information on Computer Architecture

Only a month after Burks, Goldstine, and von Neumann released their "Preliminary Discussions" in 1946, the Moore School ran its (now) celebrated summer school on building and using

Table 1. Stored Program Electronic Computers, 1950. [From Blachman (1953); Bowden (1953)]

Computer Name	Addition Time (μsec)	Tube Count	Internal Organization	Storage Size (bits)	Memory Type
Whirlwind	40	6800	Parallel	32,768	CRT
BINAC	800	700	Serial	15,872	Delay line
SEAC	850 avg	1300	Serial	23,040	Delay line
Manchester prototype	1200	~250	Serial	10,240	CRT
EDSAC	1500	4500	Serial	18,432	Delay line
ERA 1101	8500 avg	2695	Serial	393,216	Drum

electronic computers (Moore School 1946). Eckert, Mauchly, and several others on the ENIAC project gave the lectures, which covered topics ranging from arithmetic circuitry to numerical analysis. Von Neumann also lectured on the logical design of computers. One week of lectures examined in detail the electronic circuitry of the ENIAC. The lectures were attended by researchers representing the major computer research projects then underway as well as others in a position to begin a computer project.

In this early period, information about electronic computer design and construction was very hard to find. Papers describing computer research did not belong to a recognized discipline and thus might appear alongside papers about physics, radar, or applied mathematics. Much of the detailed information that was available was privately circulated rather than published. For example, the notes from the Moore School lectures were not formally published at the time, but copies of the notes found their way to many researchers. Other material was unavailable because of its security classification. However, the exchange of information about computer systems improved as the decade of the 1940s came to a close.

In the winter following the Moore School lectures, in January of 1947, Harvard University hosted a symposium on "Large Scale Digital Calculating Machinery" (Harvard Computation Laboratory 1947). The occasion afforded Harvard the opportunity to present their experiences with the electromechanical Harvard Mark I calculator (also known as the Automatic Sequence Controlled Calculator, or ASCC), designed by Howard Aiken and built by IBM Corporation (Harvard Computation Laboratory 1944). The symposium itself presented work from a broad range of researchers in the growing computer field.

An important source of early information on computer architecture was produced as a byproduct of the Whirlwind project at MIT. In late 1947, a technical report was produced that contained a complete set of block diagrams for the Whirlwind (Everett and Swain 1947). These diagrams were thoroughly annotated and described in detail the computer's logic and architecture. Unlike a similar report produced for the EDVAC (Sharpless, et al. 1946), the Whirlwind report concentrated on describing the architecture instead of the electronic circuitry. Both reports enjoyed a wide private circulation despite the restrictions imposed by military security classifications (Rosen 1969; Randell 1983).

Although security classifications restricted some of the information on computer design, there was also encouragement by the U.S. government to publish and distribute such information. The Office of Naval Research (ONR) promoted a policy of open research and information exchange, whenever possible, among computer projects it supported (Rees 1982). In 1947, ONR provided support for a graduate course on computing techniques given by Howard Aiken at Harvard University. Starting in 1949, the "Digital Computer Newsletter" was published by ONR to exchange information among participants in digital computer projects.

In 1946, ONR contracted with the staff of Engineering Research Associates (ERA) to produce a report on digital computing techniques and equipment (Tomash and Cohen 1979). With ONR's encouragement, ERA published the report in book form in 1950. The book, *High Speed Computing Devices*, made technical information about computer design and circuitry available to the general engineering and scientific communities (ERA 1950).

Performance-Driven Architecture

In March of 1951 the U.S. Census Bureau took delivery of the first UNIVAC computer (Eckert et al. 1951; Stern 1981) produced by Eckert and Mauchly, the developers of the ENIAC. Two factors played an important role in the architecture of the UNIVAC: the limited technology of the time and the UNIVAC's intended application as a business computer. Eckert and Mauchly designed the UNIVAC as a serial machine similar to the EDVAC. The main memory was built out of *mercury delay lines*, a technique that provided about ten times as much memory as otherwise could be built from a given amount of circuitry. However, the delay lines were essentially serial in nature, providing circuit economy in exchange for a lower operating speed.

Business applications were considered to be comparable to activities performed with punched card accounting equipment and thus were considered to involve far more data than could fit in UNIVAC's memory. Additional storage was provided with magnetic tape units under direct computer control. Special *buffer storage* units were also developed. The buffers were electronic storage devices that allowed UNIVAC to work efficiently with relatively slow devices such as card readers and printers. For example, a punched card

reader would transfer the sequence of characters on a punched card to its buffer at the reader's own, relatively slow, speed. When the contents of an entire card had been read into the buffer, the data would be transferred to the memory at UNIVAC's full operating speed. Thus UNIVAC did not have to waste its time slowly reading or writing individual characters to slow devices.

The UNIVAC instruction set was built to use *decimal* arithmetic instead of binary arithmetic. This decision was motivated by UNIVAC's orientation towards business problem solving: the traditional punched card accounting machines that UNIVAC was meant to supplant all used decimal notation. Many of UNIVAC's internal architectural details were generally abandoned by later designers, such as the serial organization. However, other UNIVAC architectural features, such as buffers, magnetic tape, and the decimal instruction set, found their way into later designs of business computers.

The most serious problem in building commercially viable computer systems such as the UNIVAC was the implementation of the central memory. The size and speed of the central memory was a major factor in a computer's power since it set an upper limit on both the amount of data and the number of instructions the computer could efficiently manage. The early calculators such as the Harvard Mark I (Figure 3) and the ENIAC had memories of limited size that were built out of the same elements as the rest of the calculator. The Mark I's memory used relays to store 72 words of 24 decimal digits each and the ENIAC used electronic tubes to store 20 words of ten decimal digits each. In ENIAC's case this was an expensive solution requiring over 18,000 tubes for the entire machine.

A combination of delay line memory and serial central processor design had reduced EDVAC's design to require only 6000 tubes, despite a 50-fold increase in memory size compared to the ENIAC. Eckert and Mauchly applied this lesson to Univac's design and used delay lines for the main memory. Mercury delay lines stored data as pulse sequences that circulated constantly in mercury tanks; the UNIVAC stored 1024 bits in each line. The bits were read and written serially, thus making the delay line memory fit naturally into the UNIVAC design.

Serial memories, however, did not fit well into the design of a parallel central processor; different techniques had to be found to produce effective memories for them. Most parallel memory techniques were designed around *cathode ray tubes* (*CRTs*) that could store and read back electrical charges on the surface of the tube. F. C. Williams designed an unusually effective system of this type in England, usually referred to as the *Williams tube*. Williams subsequently joined the faculty of the University of Manchester and used his tubes for storage in the Manchester Mark I computer. Williams tubes were also used in the IAS machine. J. A. Rajchmann at RCA developed another CRT based storage system called the *Selectron*, which was used in the JOHNNIAC computer developed at the RAND Corporation.

An interesting technique that combined serial parallel attributes was the *drum memory*, developed by Engineering Research Associates and incorporated into their ERA 1101 computer (Mullaney 1951). The memory consisted of a rotating drum on which data was read and written magnetically. The ERA drum contained over 200 magnetic read/write heads which could access data on parallel recording tracks as the drum spun

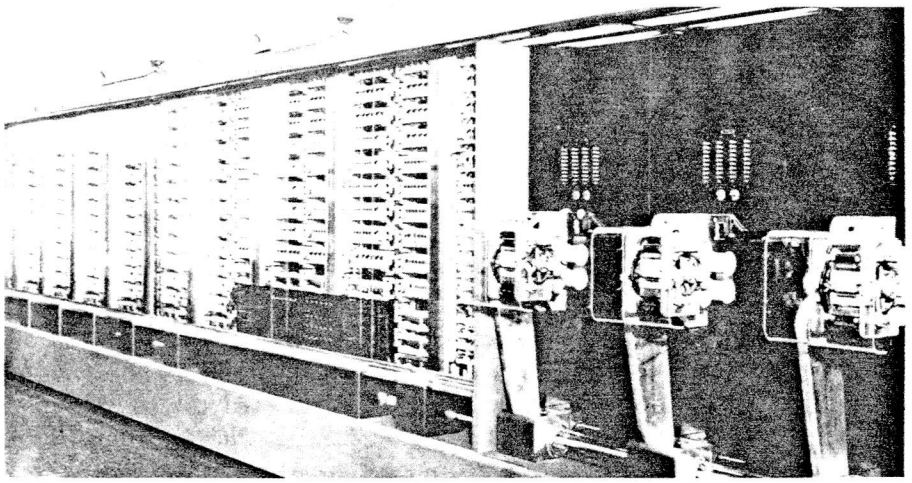

Figure 3. Harvard Mark I (ASCC).

at 3500 revolutions per minute. A drum memory was also incorporated in the Harvard Mark III, developed for the Naval Proving Ground at Dahlgren, Virginia (Poorte 1951).

The memory techniques available at that time all had problems that made them less than ideal, particularly for computers of parallel design. What was really needed was a reliable memory with a constant access time independent of the memory location being accessed. The delay lines were sensitive to temperature variations and they were fundamentally serial, thus producing a time lag while the computer waited for the delay line to circulate to a particular word. The CRT storage systems also had reliability problems, though the information generally could be accessed randomly. The magnetic drum was the most reliable of the early techniques, but it was one to three orders of magnitude slower than the underlying electronic circuits.

The primary memory problem was finally solved by the development of random access memories built out of ferrite cores. Experiments demonstrating the feasibility of ferrite core memory were performed independently and almost simultaneously by An Wang at Harvard University, J. A. Rajchmann at RCA, and Jay Forrester at MIT (Figure 4). Forrester went on to produce a working random access core memory for the Whirlwind computer in August of 1953.

The next year, in 1954, the first commercial computers with ferrite core memories were produced. The most important of these was the IBM 704, designed primarily by Gene Amdahl (Hurd 1980). IBM had been producing commercial computer systems using electrostatic storage tubes with modest success; the 704, however, was a major success and was a factor in IBM's subsequent dominance of computer sales. The 704 followed the basic architectural style of the IAS machine, consisting of a parallel central processor and 8192 words of memory organized into binary words of 36 bits each. Besides the core memory the 704 had a number of other interesting architectural features, particularly in its instruction set.

A computer's instruction set has an important impact on the computer's usefulness. To perform a computation, the computer must be programmed; the instruction set is the only language in which this can be accomplished. Initially, each computer provided a unique set of instructions. For example, the first prototype of the Manchester Mark I provided subtraction as its only arithmetic operation. Typical computers

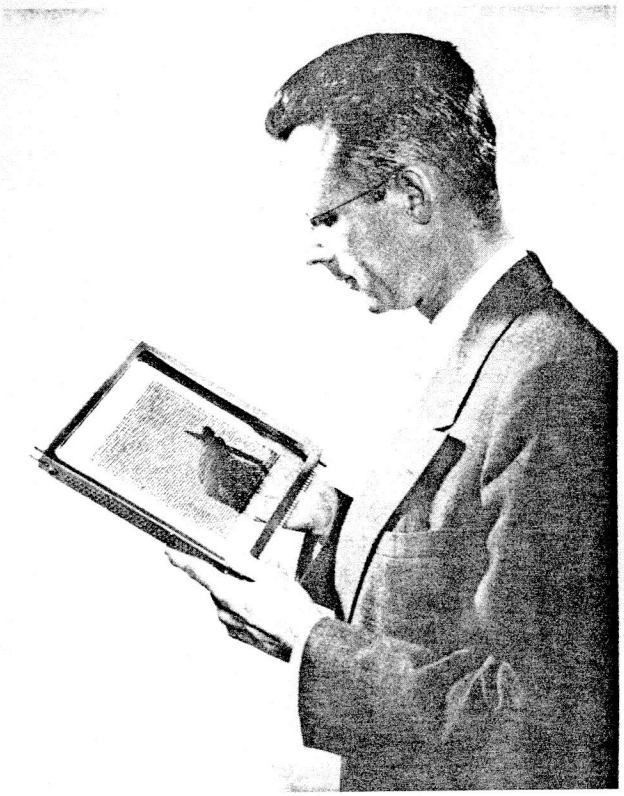

Figure 4. Jay Forrester and an early core plane.

provided three or four of the basic arithmetic operations. Programmers working with a computer lacking a needed operation would have to substitute a sequence of instructions to achieve the effect of the absent instruction. Thus, a large, well designed instruction set often simplified the work of early computer programmers.

Certain features made the 704's instruction set particularly suited for writing programs for scientific computations. One such feature was its set of *index registers*, which served the same purpose as the "B-line" introduced in the Mark I computers produced at the University of Manchester. Numerical data representing vectors or matrices are typically stored in contiguous blocks of locations in the computer's high speed memory; programs that use vectors and matrices often step through the data sequentially. Index registers greatly simplify the task of coding such problems.

Another important feature for scientific programmers was the 704's ability to compute with *floating point* numbers. Most of the previous computers could only perform computations on integer or fixed point numerical values; there was no built-in provision to manipulate numbers in

exponential notation. Numerical values in many scientific problems often exceeded the range of numbers represented on even the largest fixed point computers. This increased the difficulty of programming such problems since considerable effort went into scaling the variables so that values did not exceed the computer's range. The floating point instructions would handle the scaling automatically. Floating point instructions became a standard feature in subsequent computers for scientific problem solving.

The distinction between "business" computers and "scientific" computers became an important architectural issue in the early 1950s. Scientific machines typically operated on binary numbers and were increasingly appearing with floating point instruction sets; business machines typically operated on decimal numbers and were usually designed to accept a broader range of peripheral devices such as printers, card readers, punches, and so on. The UNIVAC operated on decimal numbers and hence was a business machine; the IBM 704 operated on binary numbers and thus was a scientific machine.

In practice, however, the architectural differences between business and scientific computers were less important to the users than had been expected. IBM was well known for its sales of "business" computers, yet many of the "scientific" IBM 704s were used for business applications, making it IBM's most profitable computer at that time. On the other hand, many of the relatively low cost decimal oriented IBM 650s (a "business computer") were purchased by universities and used for scientific calculations. The distinction became blurred as the business and scientific users adapted their problems to whichever kind of machine their organization acquired.

Two interesting architectural innovations of the 1950s were techniques to improve input/output performance. The basic problem of most devices connected to a computer is the speed differential: the computer is usually faster than the device, sometimes by several orders of magnitude. If the computer is required to slow down in order to operate the device, then valuable computer time is wasted. The original UNIVAC had buffer memories to store data being transferred between the computer and slower devices to alleviate the problem of matching speeds. The two innovations, *interrupts* and *input/output channels*, went even further towards addressing the problem of efficient input/output processing. These techniques allowed the central processor to perform independent computations at the same time that peripheral devices were inputting or outputting data.

At its research laboratory in Cleveland, Ohio, the National Advisory Committee on Aeronautics (NACA, the precursor to the National Aeronautics and Space Administration) was using a computer to control wind tunnel experiments. The computer was an ERA 1103, a model that became the UNIVAC Scientific 1103 after ERA was purchased by Remington Rand. The wind tunnel would produce input data for the 1103, the 1103 would perform computations on that data, transmit the results back to the wind tunnel, and then the 1103 would wait for more data from the wind tunnel. The 1103 could not effectively run other programs while operating the wind tunnel, even though the 1103 wasted a significant amount of time simply waiting for wind tunnel data. NACA realized that the 1103 could get more work done if it could utilize that wasted computer time.

At NACA's suggestion, the 1103 was modified to incorporate an *interrupt* facility (Mersel 1956). This gave the 1103 the ability to run another program at the same time that the wind tunnel was operating. The 1103 was provided with a special "interrupt" control signal that was connected to the wind tunnel. The 1103 was then able to execute another program. Whenever the wind tunnel needed to send data to the 1103, it would generate the interrupt signal. In response, the 1103 would suspend the other program it was running, execute the wind tunnel program, send its results back to the wind tunnel, and then resume the other program without any ill effects.

The 1103 interrupt system subsequently became a standard feature on the UNIVAC 1103A. Interrupts were also an integral part of the innovative input/output organization designed for the TX-2 experimental computer produced at MIT (Clark 1957; Forgie 1957). This system strongly influenced the design of the Ferranti Atlas computer (described in the next section) and also those built by Digital Equipment Corporation (Bell and Newell 1971; Bell et al. 1978).

Interrupt systems are particularly effective for operating input/output devices. To achieve the highest possible speed, input/output service programs must interact with their respective devices at precisely the right times. Without an interrupt system, the computer's program must spend time explicitly checking the status of input/output devices and waiting for operations to

finish. The interrupt system allows a device to alert the computer at the instant it must run the device's service program. If the device service programs are reasonably short, as they usually are, interrupts have little effect on the overall system performance.

Input/output channels are simple processors that control one or more input/output devices and share the central memory with the central processor. The central processor can direct a channel to operate a particular peripheral device, at which point the channel transfers the data between central memory and the device without involving the central processor. Channels, as well as interrupts, appeared commercially on the IBM 709 in 1958 and became a standard component of subsequent IBM mainframe computer designs (Weik 1961).

In the late 1950s, transistors began to replace vacuum tubes in computer circuitry. Several manufacturers, including Philco, Control Data Corporation (CDC), and Digital Equipment Corporation (DEC) entered the computer industry with transistorized computers. Existing manufacturers retired their computers built of tubes and replaced them with transistorized models. Some manufacturers simply replaced logic elements using the old technology with logic elements using the new, with little change to the architecture. IBM took this approach with the IBM 7090, a transistorized version of their 709 model.

Better Programming for Better Performance

Computer systems achieve their highest throughput when there is the closest match between the programs' computational techniques and the hardware on which they run. In practice, however, programmers achieve results more reliably by concentrating on the problem and not on optimizing it for a particular machine. In the late 1950s, programs that ran on one computer system seldom ran on another, even of the same model, because of differences in memory size, available input/output devices, and programming utilities.

Memory shortages in particular plagued programmers working on large problems. Programs operating on large matrices, for example, were usually tailored to the available memory size so that as much of the matrix as possible could be stored in memory by the program. Such a program could not run at all on a smaller machine or exploit additional memory available on a larger machine without major modifications. The architecture of the Ferranti Atlas, completed in 1961, made the first attempt to address some of these problems using computer hardware (Kilburn et al. 1962).

The Atlas contained two architectural features to simplify programming: "one level store," now known as a *virtual memory* or *paging* system, and "extracodes," now known as *service traps* or *SVCs*. The one level store provided a mechanism by which a program needing a large amount of memory could operate in a computer with less, even much less, central memory. Program data that would not fit in central memory was stored on a separate storage device such as a high speed disk, or in the case of Atlas, a high speed magnetic drum (Sumner et al. 1962).

A program is said to "need" a particular location in storage when one of its instructions tries to use the storage location identified by a particular memory address. The Atlas architecture added an extra level of interpretation on memory addresses generated by instructions; this interpretation would determine where the addressed storage actually was, either in memory or on the auxiliary storage drum. If the data was not in memory, Atlas would generate an interrupt and a service routine would read the appropriate data into memory for the interrupted instruction before resuming its execution.

The one level store, or paging, system afforded the programmer the same kind of abstraction enjoyed by the computer architect, who was not concerned about electronic circuitry. Instead of worrying about specific capacities of the computer's hardware the programmer could treat the computer as though it had an arbitrarily large memory. Programs written on one Atlas system could run on either larger or smaller systems without change, although there would probably be a performance penalty for using a smaller system. Programs would tend to run faster on larger systems without any revision.

The "extracode" feature also encouraged program portability. Extracodes were instructions in the Atlas instruction set that, instead of actually performing a computation, would jump to a specific subprogram in the computer's memory. The subprogram would perform the expected action. Extracodes were used to implement a variety of common utility functions such as transcendental mathematical functions, control of input/output devices, and format conversions. The extracodes

provided a standard set of functions that simplified the writing of Atlas programs. The standardization also made it easier to run programs written for one Atlas system on a different one.

While the one level store and extracodes served primarily to simplify programming, the Atlas also pioneered an important feature to improve system performance: *multiprogramming*. The Atlas system was essentially organized around a magnetic tape system for storing programs and data. A given program would typically encounter lengthy delays while data stored on tape was located. Instead of leaving the central processor idle during this interval, the Atlas supervisory software would proceed with the next program awaiting execution. Characteristically, there would be several independent programs in memory at a time, each executing in turn until blocked by a slow input/output operation. The effect is that multiple programs appear to execute at once: the waiting time of one program being exploited to execute the next portion of another.

From an architectural standpoint, multiprogramming placed two requirements on the Atlas hardware. The first was the ability to do *context switching:* the Atlas needed to be able to collect all the information necessary to save and restore the current state of a blocked program, and do so by means of its supervisory software. Since Atlas switched between programs either in response to an extracode request by the program or by an interrupt (either by an input/output device or by the one-level store system) it was able to manipulate a program's state by having the hardware save the necessary information in memory when either event occurred. Atlas contained the necessary instructions in its instruction set for restoring a program's state, thus reactivating the program at the point at which its execution was interrupted.

The second requirement, *program protection*, was brought about by the need to debug programs. Since Atlas relied on special software to manage the system (the Atlas Supervisor) as well as on special hardware to do program switching, the system needed to prevent errors in users' programs from damaging the Supervisor. Otherwise, a serious error in one user's program could have interfered with or even damaged other users' programs.

The Atlas hardware executed users' programs in a special mode that restricted the program's ability to read and write memory or to affect the input/output devices. When a user program executed an extracode or when an interrupt occurred, the Atlas would switch to an unrestricted mode that allowed complete access to the system's resources and capabilities; all access to shared system hardware by user programs was accomplished through extracode operations.

A significant aspect of the Atlas' hardware innovations was that their usefulness relied on a specific piece of software: the Atlas Supervisor (Kilburn, et al. 1961; Howarth, Payne, and Sumner 1961). The hardware innovations supported the sharing of the computer's hardware among several programs concurrently; the Supervisor provided controlled access to the shared facilities and prevented their misuse. Prior to this, features of a computer's architecture were designed to be used by any programmers who needed to use them. The Supervisor used the program protection features to prevent access to the paging and multiprogramming hardware by other programs. The Supervisor provided extracode instructions that other programs could use to request additional memory or perform other actions that involved shared resources. This provided a reliable mechanism for sharing global resources and generally simplified the programming task.

In 1963, a group of hardware and software designers at Burroughs Corporation produced the B5000 system (Lonergan and King 1961; Rosin 1987). The designers of the B5000 took ease of programming to a new level in the B5000 architecture by designing the computer to be programmed entirely in a high level language, ALGOL. Prior to the B5000, computer architects had assumed that programs would primarily be written in the instruction code peculiar to that computer, usually referred to as the *machine language* or *assembly language*. These languages would describe computational procedures as a sequence of specific operations on the computer's storage registers and memory. However, progress had been made in developing *high level languages* in which computations could be described in various pseudo-mathematical notations. Several languages had been developed by 1960, including FORTRAN, LISP, and COBOL (Wexelblat 1981). ALGOL was an important high level language developed at that time by a team of computer experts in Europe and the United States.

By building the B5000 system around ALGOL, the designers managed to exploit the advantages of high level languages and overcome some of the disadvantages. The major advantages were that high level languages were easier to learn and use

than machine languages. A few statements in a high level language would often express the same computation as an entire page of machine language, and express it in a far more readable form. Programs written in high level languages were also more "portable" than machine language programs in that they could often be used on other computers without having to be translated into a different language.

The disadvantages of high level languages were generally considered to be related to performance: such programs often used more computation time and memory than comparable programs written in machine language. This performance penalty arose from the poor quality of the process of translating the high level language program into the computer's machine language. The translation was performed by a special program, called the *compiler,* whose performance depended on the skill of its creators and on how closely the expressive capabilities of the high level language matched the computational abilities of the computer's machine language. Both of these competencies were present in the development of the prototype FORTRAN compiler for the IBM 704 (Backus 1979; Allen 1984). However, ALGOL had not been designed with a specific computer hardware in mind, so the Burroughs designers tried to fit the hardware to the language.

By basing the system design on ALGOL, the B5000 achieved an instruction set and memory organization that was completely different from other computers. To implement ALGOL efficiently, the B5000 contained special architectural features: *relative addressing, tagged memory, stacks,* and an elaborate instruction for calling *subroutines.* Efficient subroutine handling was essential to the efficient execution of ALGOL programs and the use of stacks was an important innovation.

The first two features simplified the management and use of the B5000's memory. *Relative addressing* eliminated the need to assign specific storage locations to a program until the program was actually executed, simplifying the task of allocating computer memory. In the *tagged memory,* additional information (the "tag") was affixed to every memory word to identify the kind of data in the word, independent of the actual data value stored in that word. In other computers there was no way to determine if a given memory word contained an instruction or data, since both were often represented by identical binary patterns. The tag on a B5000 memory word told whether the word contained an instruction, data, a subroutine parameter, an address pointer of one kind or another, or other control information required by the system. The tag words allowed the system to treat different words differently according to the type of information in a word. For example, ALGOL contained three different methods of determining the value of a subroutine's parameter; the B5000 acquired a parameter's value according to the type of parameter as given by the tag word.

Since the beginning of computer programming, it has been a common practice for programmers to produce a large program by combining a number of smaller programs, called *subroutines.* The technique was suggested in the IAS Report (Burks, Goldstine, and von Neumann 1946) and described in practice by the developers of the EDSAC (Wilkes, Wheeler, and Gill 1951). Each subroutine could be a small program that solved one aspect of the problem or performed a common computation. The "main" program would then *call* the subroutines individually to perform their respective actions. Each subroutine would then in turn call the subroutines it needed to perform parts of its computation, and so on. Programs in most high level languages, including ALGOL, could be written as a group of subroutines.

In a computer, a *stack* is a storage area in which items of data are stored on a last-in first-out basis. A stack provides a simple way of saving information used by one subroutine when it calls yet another (Randell and Russell 1964). The B5000 subroutine call instruction automatically saved necessary information on the stack, acquired the parameters required by the subroutine being called, and then called the subroutine. ALGOL programs running on other computers might require over a dozen machine language instructions to perform a complicated subroutine call, whereas the B5000 required only one instruction.

Another important B5000 innovation was a variant of the concept of multiprogramming: *multiprocessing.* Multiprocessing systems contain two or more central processors and use them simultaneously for executing programs. B5000 systems could contain one or two processors; both could be used to execute user programs. The supervisory program would give a user program to each processor to execute and the two programs would then be able to execute simultaneously. Like the Atlas, the B5000 series relied heavily on its supervisory program, the MCP, to provide many features inherent in the computer's architecture.

Although the B5000 pioneered the ALGOL and

multiprocessing features, Burroughs' success resulted from the higher performance B5500 system which appeared shortly thereafter. Today, Unisys (the present corporation descended from Burroughs) still produces computer systems that execute programs written in 1963 for the earliest machines in the B5000 series. The notion of newer hardware being compatible with older software was beginning to be taken seriously at the time; Burroughs' success is a tribute to the notion of designing the instruction set around a high level language.

Software compatibility became a major issue in computer architecture when the IBM System/360 family of computers appeared in 1965, one year after its announcement (Amdahl, Blaauw, and Brooks 1964; Blaauw and Brooks 1964). A collection of computers constitute a *family* when they maintain a high degree of hardware and software compatibility. In a well designed family of computers, components of one computer can generally be used with any other member of the family, regardless of whether the component is a software program or an input/output device.

After a careful examination of trends in the computing industry (Haanstra 1961), IBM decided to concentrate its efforts on the System/360 computer family. Software compatibility for the System/360 implied that all systems would support the same machine language instruction set. Hardware compatibility implied that input/output devices built for the System/360 family should be able in general to be connected to any model of the System/360.

As a family, all System/360 computers shared several architectural attributes (IBM 1970). All were parallel computers with binary arithmetic, input/output channels, and a sophisticated interrupt system. All processors supported the same basic instruction set, including the *SVC* instruction which performed the same function as the Atlas extracodes. All programs written by users of System/360 computers would use the SVC instruction to perform "supervisor" functions such as input/output operations. The System/360 also supported two additional instruction sets as options: a "commercial" (decimal) instruction set and a "scientific" (floating point) instruction set.

The System/360 family provided a variety of compatible computers over a range of cost and performance characteristics. IBM achieved a high degree of compatibility among the System/360 computers and those of descendent families, including the System/370 (IBM, 1974), 30xx, and later models. Compatibility over such a long range of time and technology represented an impressive architectural achievement, though it perhaps limited IBM's opportunities for further architectural innovation.

The Rise of Supercomputers

In the late 1950s there were major efforts to improve computer performance through architectural innovation. In response to a contract from the Los Alamos Scientific Laboratory, IBM began Project Stretch, an attempt to build a new computer that would "stretch" computing technology and produce a hundredfold improvement in computational performance (Dunwell 1956; Buchholz 1962). The Stretch computer embodied every architectural innovation at IBM's disposal, including transistorized circuitry, channels, interrupts, and index registers. It also pioneered the concept of *instruction prefetching* to improve the speed at which Stretch executed instructions. Stretch would try to collect instructions from memory before they were needed (i.e., "prefetch") and partially interpret them at the same time it was executing the current instruction (Bloch 1959).

Stretch gave IBM valuable experience with advanced architectural ideas and the machine provided Los Alamos with a significant increase in computation power. However, the machine itself was a commercial failure. Stretch did not come close to a hundredfold improvement in performance that had been promised and IBM had to sell the machines at a significant loss. Improvements in core memory technology allowed IBM's customers to get about half of Stretch's performance from the newer IBM 7094 at a far lower cost.

Prior to IBM's initiation of the Stretch project, UNIVAC had begun its own project to develop a high performance computer, the LARC (Eckert 1956; Eckert et al. 1959). Like Stretch, LARC's circuits were designed with transistors, but LARC's earlier start saddled it with slower transistors of older design. Also like Stretch, LARC failed to achieve its lofty performance goals, but it did pioneer a concept incorporated in future high performance computers: *interleaved memory*. LARC was designed to operate with a group of memory banks, all of which could be read or written to independently. When reading a sequence of data words from four interleaved memory banks, four words at a time can be read from memory simultaneously. While the first word of the sequence is being read from the first memory bank,

the second, third, and fourth words would be read from the second, third, and fourth memory banks. Thus, the memory speed increased in proportion to the amount of interleaving when reading vectors of data from memory.

In 1964, Control Data Corporation (CDC) introduced the CDC 6600 computer, Figure 5, (Thornton 1964; 1980), the fastest computer at the time and for several years following. Designed by Seymour Cray, who had earlier designed the CDC 1604, the 6600 essentially took the opposite approach to that of the Ferranti Atlas three years earlier. Instead of emphasizing ease of programming, every effort in the 6600's design went into organizing the computational components so they could work together at the highest possible speed.

An important feature of the CDC 6600 central processor was that it reintroduced the concept of multiple *functional units*. A functional unit in the 6600 was a computational circuit built to independently perform a major arithmetic operation such as addition or multiplication. The program could exploit the presence of several functional units so that the central processor would do several arithmetic operations in parallel, thus improving the speed of computation. The concept had been used on the ENIAC, but had never before been applied to a stored program computer design.

An important innovation was in the 6600 instruction set (CDC 1968). Unlike previous computer instruction sets, the 6600 did not use explicit "load" and "store" instructions to transfer data between central processor registers and memory, but rather it had a set of address registers with corresponding data registers. Whenever an address was placed in an address register, the data at that address would be read from memory and placed in the corresponding data register. Changes to the contents of a data register would cause its contents to be stored back in memory. These transfers could occur independently of instruction execution, offering an additional improvement in speed. With the appearance of the CDC 6600 FORTRAN compiler, programmers could exploit the 6600's architectural innovations without having to understand its novel architecture.

It was several years before the 6600 saw serious competition for its place as the fastest computer. The ILLIAC IV, a multiprocessor project of the late 1960s, was a significant experiment in computer architecture (Barnes et al. 1968; Bouknight et al. 1972). The ILLIAC IV was designed by researchers at the University of Illinois and built by the Burroughs Corporation. It consisted of 64 independent minicomputers coupled together so that they could simultaneously execute a single program that performed a single, large computation. The minicomputers were arranged in a two-dimensional matrix and could transfer data words rapidly between neighboring minicomputers. The design was particularly suited to performing matrix computations.

Although the ILLIAC IV could perform some

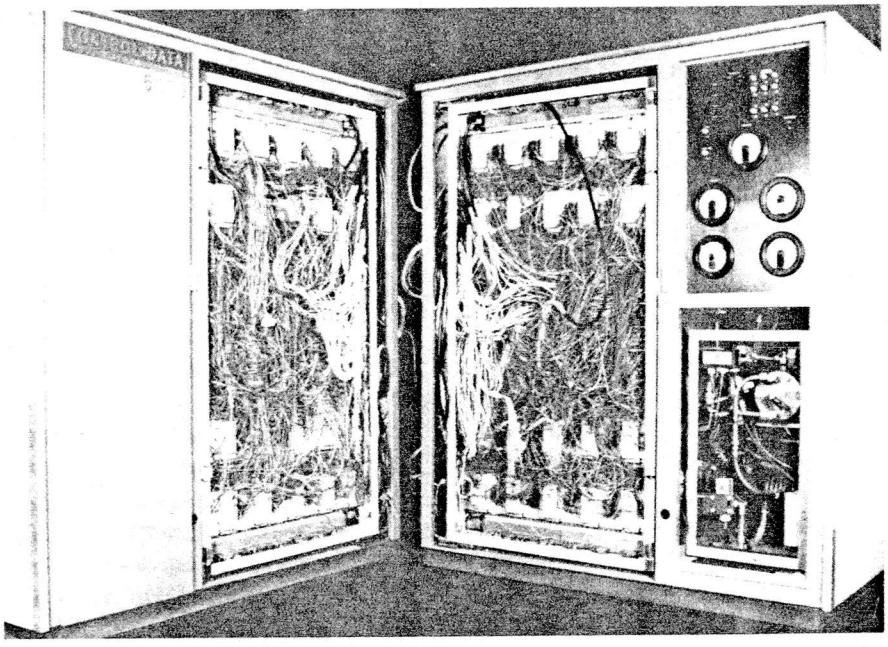

Figure 5. Control Data 6600.

computations much faster than the 6600 it was not a major success, primarily because it required special programming. By the time the ILLIAC IV appeared, typical users of high performance computers were writing all of their programs in FORTRAN and were not willing to learn how to program in a complicated assembly language such as that of the ILLIAC IV. Users who needed and could afford a high performance computing system were usually experts in a computationally intensive problem domain such as physics or applied mathematics and not in machine language programming or computer architecture. Typical programs written in FORTRAN could not automatically exploit the ILLIAC IV's 64 processor parallelism.

The relative success and acceptance of the CDC 6600 in comparison to the ILLIAC IV marks the genesis of the modern *supercomputer:* a high performance computer system for solving numerical problems coded in a high level language, usually FORTRAN (see Sammet 1987). Supercomputer users wanted to concentrate on solving their problem and not on fitting the problem to a special computational architecture, regardless of its performance. FORTRAN's relative portability and widespread use made it a basic requirement of any effective high performance computing system. Except for special purpose projects, subsequent high performance computer systems concentrated on improving the performance of FORTRAN programs.

In 1971, Texas Instruments produced a supercomputer called the Advanced Scientific Computer, or ASC (Watson 1972; Dean 1973). It was a true *vector* computer because it achieved its highest performance when operating on sequences (or vectors) of numbers. The instruction set contained special instructions for specifying and operating on vectors of numerical data. The ASC's central processor had two separate arithmetic units that could be set up to operate concurrently. The memory system contained several banks of interleaved high speed memory from which vectors could be read with a high degree of parallelism.

The circuitry of the arithmetic units also used *pipelining*, a technique that increased computational speed when working on a sequence of data words. The technique bore similarities to the concept of instruction prefetching used earlier in Stretch, but applied to data instead of instructions. Pipelining exploited the fact that a single arithmetic operation on a single number or pair of numbers usually required the arithmetic unit to perform several separate steps. In a pipelined arithmetic unit, separate computational circuitry and intermediate data registers were built for each step. When the arithmetic unit performed an instruction on a vector of data, the arithmetic unit worked on several data words at once in an assembly line fashion. Once the arithmetic unit finished the instruction's first step on a data word, it would start the first step on the next data word in the vector as the second step took place on the previous word.

Significantly, the ASC could be programmed in a standard dialect of FORTRAN. Although the ASC performed well with computations on long vectors, it performed poorly with scalar computations because of the amount of overhead encountered when setting up the arithmetic units and fetching data from memory. This overhead was minor when operating on long vectors, but greatly reduced the ASC's performance on scalar computations.

In 1976, Seymour Cray produced the Cray-1, a high performance vector processing supercomputer (Russell 1978). As he had with the CDC 6600, Cray built the fastest machine of its time. Unlike the ASC, the Cray-1 was fast at scalar computation as well as vector computation, so it was a faster computer overall than the ASC. Performance on vector arithmetic was improved with a set of *vector registers*. As in the CDC 6600, all arithmetic operations operated solely on data in central processor registers. The vector registers provided a special bank of registers to hold entire vectors of numbers for fast computation. Thus, the Cray-1 could perform arithmetic operations on entire vectors without having to go through the central memory.

The Cray-1 did not use multiple arithmetic units like the ASC, but instead contained multiple functional units as did the CDC 6600. A functional unit would typically perform a single arithmetic operation such as addition or multiplication while an arithmetic unit would usually provide a complete set of arithmetic operations. Since they were specialized, functional units were usually less complex and costly than arithmetic units of comparable performance. Thus, functional units provided more performance at a given cost.

The Cray-1 was also designed physically for speed: the system itself was much smaller than competing high performance computer systems. A Cray-1's central processor required approxi-

mately 38 square feet of floor space compared to over 100 square feet for the competing CDC 7600. Computer systems had reached the point where signal speed along the interconnecting wires had become a major constraint on the computational speed. The Cray-1 was designed around a compact wired backplane arranged in a $^3/_4$ circle, allowing shorter wire runs and correspondingly shorter signal delays. As was the ASC, the Cray-1 was provided with a special FORTRAN compiler that allowed standard FORTRAN programs to exploit the vector architecture features.

Cost-Driven Architecture

Perhaps the most striking feature of computer economics is the rapid and continuous drop in the cost of the underlying electronic circuits. Over the past quarter century, the cost of digital logic circuitry has dropped by approximately 30 percent every year (Bell, Mudge, and McNamara 1978). This has been important in the development of higher performance systems since it has made increasingly complex and faster circuitry economically feasible. This drop in cost has also reduced the price for the least expensive computer circuitry. The lowering costs have combined with a corresponding drop in circuit size, allowing computer control to appear in just about any application.

The importance of cheaper logic is particularly striking in light of *Grosch's Law*, a celebrated piece of folklore about the economics of computer design (Knight and Cerveny 1983). In the late 1940s, Herbert R. J. Grosch perceived a simple relationship between a computer's performance and its cost, namely that the performance increases as the square of the cost. See Figure 6 (Ein-dor 1985). Thus, a fourfold improvement in computing power should double the cost of the computer. The law became widely known and quoted, despite the fact that Grosch himself never published it. The implication for low cost computing was clear: architectural tricks could not lower the cost of a basic computer; low cost computing had to wait for low cost logic.

The original customers pursued by computer manufacturers were those with business data processing problems or those with science or en-

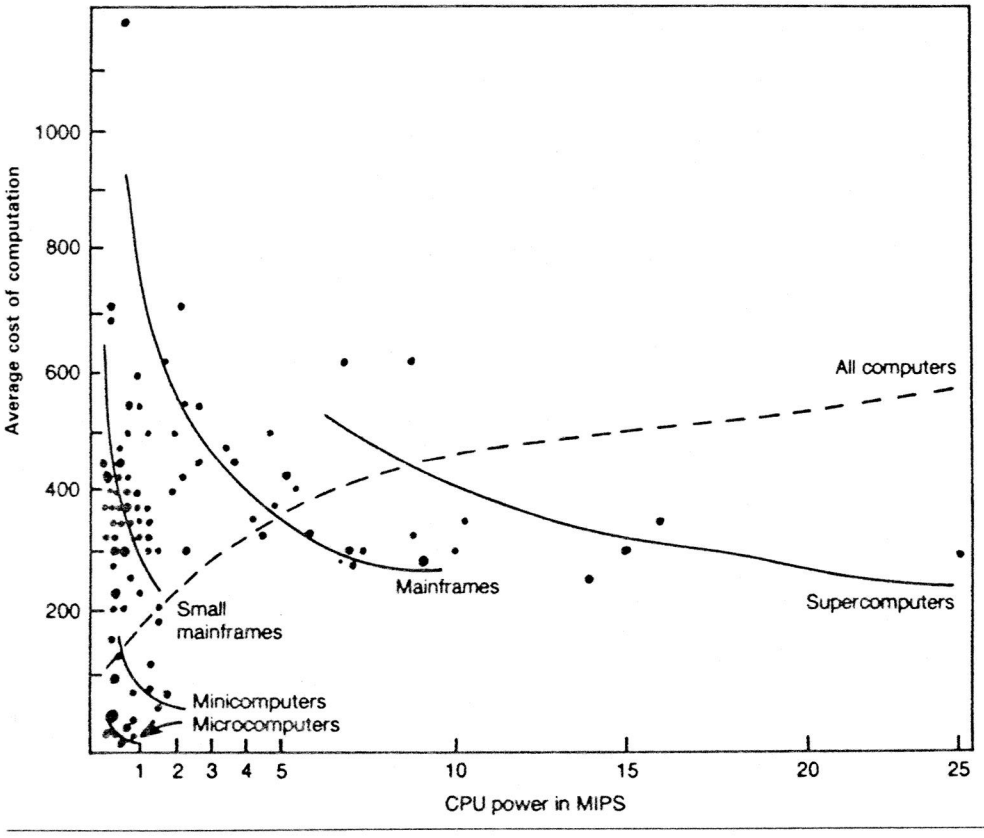

Figure 6. The relationships between power and efficiency for the five classes of computers. Reprinted from Ein-dor 1985 (Copyright 1985, Association for Computing Machinery, Inc., reprinted by permission).

gineering problems, both of whom willingly paid for as much computational power as could be made available. However, the computer clearly had potential in applications requiring flexible automatic control. Such problems could be addressed by a more austere computer design: one with a low cost, a simple instruction set, a small memory, and enough speed for their application. As electronic technology improved, the falling cost of logic circuitry justified the production of austere computer designs.

Earlier it was said that improving technology does not necessarily improve computer architecture. In a sense, an austere computer design requires that the designer go back to older techniques that use less logic circuitry. Researchers in computer architecture have recognized a "wheel of reincarnation" in which simple techniques from the early days of an older technology get reused to produce low cost systems in a newer technology (Myer and Sutherland 1968). Between 1965 and 1977, for example, minicomputers acquired sufficiently complicated architectures to be almost indistinguishable from computers classified as mainframes (Leonard 1987). The same phenomenon is today in effect for microcomputers.

In practice though, major milestones in this quest for economy have involved architectural innovations. The less expensive computer systems have widened the range of cost-effective computer applications. In turn, these new applications have occasionally fueled architectural improvements to enhance a computer's performance in carrying out those applications. For example, it is far more common to find flexible, high performance graphics systems, data communications systems, and interactive programming systems on the smaller scale computers. These were applications that were seldom economically feasible on expensive computer systems and in many cases they could not be accommodated by the channel oriented input/output systems of most mainframes. Smaller scale systems tended to have more flexible input/output systems that were often designed to support such applications.

The first computer that was clearly designed to be of low cost was also the first computer to successfully execute a stored program: the first prototype of the Manchester Mark I (Williams and Kilburn 1951). The computer was developed by F. C. Williams and Tom Kilburn at the University of Manchester, England, and used *Williams tubes* for its memory system, as described earlier. Williams and Kilburn were primarily interested in building a reliable digital storage system and this "baby" Mark I was designed and built to test the prototype memories. This goal made the prototype different from other computers built at that time: instead of emphasizing arithmetic, the design emphasized programmable control and circuit economy. The design eliminated many of the arithmetic operations appearing in other computers, thus reducing the amount of circuitry.

The decision by Williams and Kilburn to build a computer instead of an elaborate testing circuit illustrates the importance of even a minimal computer: flexibility and automatic control. The designers would have had a hard time designing and building a memory tester that could test the memory as thoroughly as even the simplest computer, and the computer was probably as easy to design and build. The prototype's simple design was perfect for the memory testing task and demonstrated that a computer with an austere design can be of value in situations requiring automatic control.

The prototype Mark I incorporated several architectural features that provided the designers with an economical but useful machine. First, they started with the simplest instruction set of any stored program computer of that time. The prototype had only seven instructions and the only arithmetic instruction was "subtract." This was sufficient for memory testing and, though inconvenient at times, sufficient for many other programming problems. The prototype's hardware fit onto eight tall equipment racks, making it a small fraction of the size of contemporary computers such as the ENIAC or Whirlwind. To further simplify the circuitry, the designers used Williams tube memory wherever possible to store intermediate data used when fetching and executing instructions. Additional savings were achieved by using a serial circuit to do subtraction instead of a larger parallel circuit.

Few companies manufactured low cost computer systems in the 1950s. Successful low cost systems were manufactured by Bendix (the G-15) and Librascope (the LGP-30) at a cost of approximately $50,000 for basic configurations. The successful IBM 650 (Hurd 1986) was also a low cost computer; at a minimum cost of $200,000 it was significantly less expensive than IBM's other computer systems (Figure 7). These systems all achieved low costs by using rotating drums for central memory and serial arithmetic circuitry. The drum memories and serial circuits lowered manufacturing costs but yielded lower operating

Figure 7. IBM 650 data processing system.

speeds. These low cost systems brought automatic computation to organizations that could not afford a faster computer.

However, the low cost systems of the 1950s were too slow for time-critical applications. The systems with rotating drum memories often required an average of two milliseconds to add two numbers: a hundredth the speed of an IBM 704. But the faster systems were often too expensive to be used for automatic control problems. Such applications usually required the all of the computer's attention and prevented its use for other data processing problems.

There were very few real time computer applications in the 1950s in which the required performance justified the cost. The principal example was the SAGE air defense system, a network of AN/FSQ-7 computers (Figure 8) that used radar sightings of airplanes to direct interceptor fighters (Everett 1983). The AN/FSQ-7 computer was based on the Whirlwind (Everett 1951) at MIT and was manufactured by IBM. The cost of using vacuum tube computers for real time control was justified by the importance of the system to national security. The widespread use of computers for real time control had to wait for lower cost computer systems that could be dedicated to their tasks.

Minicomputers

There were essentially no manufacturers of fast, inexpensive computer systems until Digital

Figure 8. SAGE air defense system consoles.

Equipment Corporation (DEC) introduced the Programmed Data Processor (later called the PDP-1) in 1960 (Bell et al. 1978). The PDP-1's simple design, coincidentally, descended directly from a series of memory and circuit testing computers built at MIT and Lincoln Laboratory in the 1950s: the TX-0 (Mitchell and Olsen, 1956) and the TX-2 (Clark, 1957). A single PDP-1 sold for about $110,000, which was less than the cost of an input/output channel for an IBM mainframe computer of the same period (Weik, 1961). It could add two numbers in five microseconds, making it orders of magnitude faster than other computers in its price range at the time. For many reasons, architectural as well as economic, the PDP-1 opened a new realm of computer applications.

The PDP-1's design achieved both high speed and low cost by well chosen tradeoffs. This is illustrated in Table 2. The PDP-1 used features of more expensive computers such as ferrite core memory and parallel arithmetic circuitry instead of the rotating drum memory and serial circuitry of other low cost systems. The instruction set was very simple, containing only 25 instructions, compared to 91 instructions on the IBM 704. The small number of instructions reduced the complexity of the instruction decoding circuitry.

Significant savings were also achieved by using relatively small *word size*. The word size determined the range of numbers that fit in a single data word; a word being the unit of data with which the computer could work conveniently. Computer memories and arithmetic units were designed to efficiently handle data words of a particular size, specified by the number of binary or decimal digits contained in a single word. The PDP-1 used 18-bit words compared to IBM 704 which used 36-bit words.

In a parallel architecture, the size of the arithmetic, data transfer, and data storage circuitry were all proportional to the computer's word size. The PDP-1's smaller word size required less circuitry, allowing it to be produced at a lower cost. The small word size made the PDP-1 less convenient for large scale computation problems, but its low cost made it affordable. In practice, a larger word size was unnecessary for many applications, particularly those involving real-time control.

The PDP-1's architecture included a flexible input/output system that was very effective in communications and real-time control applications. Special features of the input/output system included a high performance interrupt system and the ability of input/output interface circuits to transfer data directly to or from memory. This permitted high speed devices such as magnetic tapes or disk storage systems to be attached at a relatively low cost. The design also supported the use of extremely simple interface circuits for slow devices such as teletypes. This made the PDP-1 practical for use in early message switching and interactive computing projects. The input/output system also provided an excellent interface to the PDP-1's built-in CRT display.

Fast data transfer operations took place using a technique known as *direct memory access,* or *DMA*. An input operation would start with the PDP-1's program instructing the appropriate device to write a specific number of data words into a specific location in central memory. Each time the input device was ready with a word of data it would tell the PDP-1 to write the word into the next location in memory. The PDP-1 would do so immediately, in effect "stealing" a memory cycle from the program being executed. After all of the words are transferred, the device would produce an interrupt to signal that it is finished. Output operations worked in the same way, but in the opposite direction.

Manufacturers of data processing systems, particularly IBM, had very strict policies against modifying their hardware or attaching custom

Table 2. Selected Electronic Computers, 1961. [From Weik (1961); Bell et al. (1978)]

Computer Name	Addition Time (μsec)	Cost per System	Word Size	Memory Type	Number of Instructions
IBM 7090	4	$3,630,000	36	Core	187
PDP-1	5	$110,000	18	Core	25
IBM 704	24	$1,994,000	36	Core	91
IBM 709	24	$2,630,000	36	Core	187
IBM 650	2736 avg	$250,000	~34	Drum	89
Librascope LGP 30	5500 avg	$55,860	32	Drum	16
Bendix G-15	7520 avg	$77,300	29	Drum	125

built input/output devices. These policies were a serious problem for customers experimenting with computer control or with unusual input/output devices. DEC took the opposite approach with the PDP-1. Systems could be purchased with custom input/output device interfaces for controlling special devices. DEC also provided the technical information necessary for customers to build their own control circuits. For example, researchers at MIT and at Bolt, Beranek and Newman modified PDP-1 computers to provide support in the computer hardware for multiprogramming experiments. In other cases, the systems were used to control custom-built equipment or laboratory experiments, or to handle data communications. As seen in Figure 9, a PDP-1 was interfaced to the controls of a model railroad system (Lee 1985).

In 1965, DEC introduced the PDP-8 at approximately a tenth of the cost of a PDP-1 (Bell and McNamara 1978). Its producers called it a *minicomputer*, a new and economical source of computing power. The PDP-8 had a shorter 12-bit word length, but it performed as fast as a PDP-1 and its input/output system was equally flexible. Although the PDP-8 was economical and a successful product, it was difficult to program—primarily because of its limited *address space*. The size of a computer's address space indicates the amount of memory that instructions can easily address. PDP-8 instructions could only access data within 128 word sections of its central memory. Accessing data in other sections of memory was cumbersome and still could not address more than 4096 words.

By 1970, improvements in integrated circuitry made it feasible to produce a computer with a larger address space and better instruction set at the same cost as the original PDP-8. This led DEC to produce the PDP-11 computer (Bell et al. 1970) with an address space of over 32,000 words. The PDP-11's instruction set combined several simple ideas to produce an extremely flexible set of *addressing modes* for accessing data. The addressing modes were built around the use of eight registers in the central processor: the program counter (containing the address of the next instruction to execute), the stack pointer (used for manipulating a stack in memory), and six general purpose registers. These registers were accessed by means of eight *modes* which provided the customary methods of addressing data as well as *relative* and *stack* addressing similar to that provided in the Burroughs B5000. The addressing modes gave the PDP-11 an instruction set far superior to most other computers of its time.

Another noteworthy architectural feature of the PDP-11 was its unified strategy for manipulating memory and input/output devices. Input/output interface circuitry was typically designed with internal registers containing control and

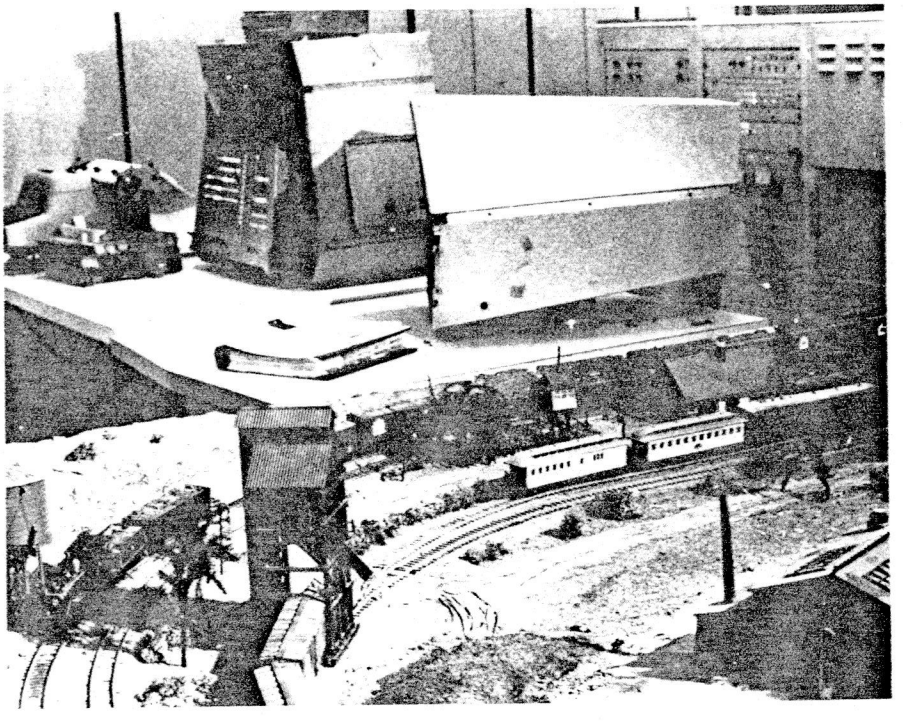

Figure 9. PDP-1 and a model railroad at the University of Massachusetts.

status information; these registers were accessed via input/output instructions on other computers. On the PDP-11 these registers were assigned memory addresses and the corresponding input/output devices were controlled by writing appropriate data to the interface's registers. The PDP-11 thus did not have to have a special set of input/output instructions; any instruction that manipulated data in memory could be used.

This uniformity of the PDP-11's addressing structure was also reflected in the hardware. Both memory circuits and input/output interface circuits were connected to the PDP-11 via its Unibus, a fast, flexible, and well defined parallel *bus*. Many manufacturers, particularly DEC, designed their input/output systems around a bus: a set of high speed connections to which input/output interface circuitry could be attached. An important innovation of the Unibus was that memory circuits as well as input/output interfaces could be attached to it interchangeably. A Unibus connection could be used either for adding memory or for adding an input/output interface to the computer depending on the customer's needs.

Microcomputers

Although the PDP-11 represented an advance in cost-driven computing, it did not represent an advance in economy. Competing manufacturers produced less expensive and equally fast or faster machines by utilizing the continuous fall in the prices of electronic components. The next major step in austere computing came the following year, in 1971, when Intel produced the Intel 4004, the first microprocessor (Morse et al. 1978). Large scale integration had produced an entire central processor in a single integrated circuit package.

The Intel 4004 represented a major advance in computer architecture, primarily because a central processor could suddenly be treated as a predefined building block in a computer design. This opened computer design to a broader range of designers by eliminating a large part of the work. The computer designer no longer had to design the arithmetic unit, instruction set, and associated logic. By accepting the constraints of a particular microprocessor's word size, instruction set, and interrupt structure, the designer could concentrate on optimal or economical circuits for memory and input/output control.

The 4004 alone did not change computer design: it was only the first of the microprocessors. In fact, its four-bit word size restricted it primarily to use in handheld electronic calculators. The first widely used microprocessor systems were produced using later microprocessors such as the Intel 8080 and the MOS Technologies 6502 with their larger eight-bit word size. The Intel 8086/8088 microprocessors with their 16-bit arithmetic instructions and the Motorola 68000 with its 32-bit arithmetic instructions subsequently became the microprocessors of choice in small computer systems. Both of these have fathered families of microprocessors that are widely used today.

In 1974, the TMS1000 monolithic microcomputer was introduced by Texas Instruments. It was a complete computer system on a single piece of silicon, containing the central processor, input/output circuits, and permanent and read/write memory (Texas Instruments 1975). The TMS1000 was the first of a family of such microcomputers, all with the same basic architecture but with varying instruction sets, memory sizes, and input/output capabilities. Originally used in pocket calculators, TMS1000 family computers have appeared in numerous other applications. By 1979 over 35 million of them had been sold, making it the most widely used computer.

The rapidly falling cost of computer circuitry brought about unprecedented growth in the use of computers. The TMS1000 represented part of this growth: a hidden revolution in which simple analog and electromechanical controllers were replaced with microcomputers. More visible growth is represented by the personal computer. After the appearance of the microprocessor, many individuals and a few small firms tried to produce low cost systems for general purpose computing. The first significant personal computer was the Altair 8800 (Figure 10) which appeared in 1975 (Warren 1977). The Altair 8800 is often credited with spawning the personal computer industry and it represented a successful early attempt at personal computer design (Levy 1984).

Despite its early success, the Altair 8800 did not dictate the direction of microcomputer architecture. The Altair was a computer built in the tradition of earlier minicomputers: it came without built-in input/output facilities for human communication such as a keyboard, printer, or video screen. The keyboard and video display interface for the Altair were built on separate circuit boards and attached to its input/output bus, called the S-100 bus. Without such a circuit board the Altair had only a bank of switches and lights for reading and writing binary data in memory.

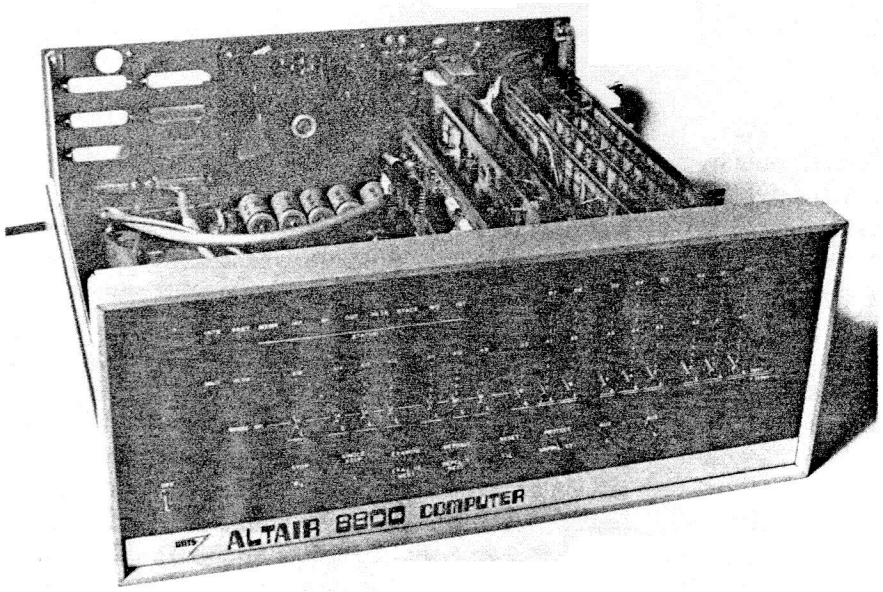

Figure 10. MITS Altair 8800 computer.

One or more additional boards were required to provide the central memory. The multiplicity of separate circuit boards made the Altair more expensive than subsequent personal computers that incorporated these circuits on a single circuit board. The Altair's multiple circuit boards significantly raised costs above those of competing systems while providing flexibility that was unnecessary for most personal computer users.

The Altair 8800 was soon supplanted in the personal computer market by computers with built-in keyboards and video interfaces such as the Apple II which appeared in 1977 (Wozniak, 1977). The Apple II was built around the eight-bit 6502 microprocessor and although it did not use the S-100 bus, it did have a bus of its own design for attaching additional memory and input/output circuits. Today, the Apple II is still widely used in home and educational environments. Its built-in keyboard and standardized bus for adding optional circuits have become the basic elements of the personal computer.

One of the most important features of the Apple II was *bit mapped graphics*. The Altair had followed computer industry traditions by assuming that users would communicate via a teletypewriter-like terminal. However, alternative techniques that exploited elaborate graphics were under development. The Alto system (Figure 11), developed and used by researchers at Xerox in the early 1970s (Thacker, et al. 1979), was an experimental personal computer that used its bit mapped graphics capability to display text and drawings simultaneously to its user. Programmers on the Apple II adopted some of these techniques successfully despite the Apple II's lower performance. The concepts behind the Alto are most evident in modern *workstations* (ACM, 1986)

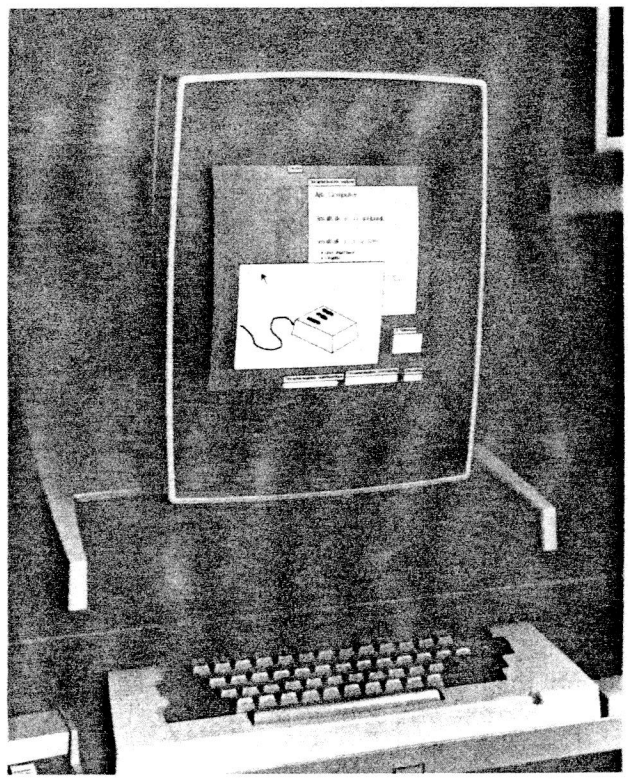

Figure 11. Xerox Alto computer system (reproduced with permission of the Xerox Corporation).

such as the Apollo (Apollo 1981) and Sun systems and Apple's Lisa and Macintosh systems (Baecker and Buxton 1987).

The IBM-PC, introduced in 1981, continued and enhanced the personal computer architecture that made the Apple II successful (IBM 1981). The IBM-PC was built around a single circuit board containing a central processor, keyboard controller, and a bus for adding memory, a video controller, and input/output devices. Separate video controllers provided bit mapped graphics with different resolutions and color capabilities. For its central processor, the IBM-PC used the Intel 8088 microprocessor containing instructions for manipulating eight- and 16-bit words. Today, the architecture of the original IBM-PC is the standard architecture for a basic personal computer and is widely copied.

In the late 1970s a group of researchers at the University of California, Berkeley, initiated a controversy in computer architecture by discussing the potential advantages of a *reduced instruction set computer*, or RISC (Patterson and Séquin 1982). The RISC advocates argued that the widespread use of high level languages such as FORTRAN, PASCAL, and COBOL make an elaborate instruction set unnecessary from a programming standpoint. They argued further that a computer with a simple, but fast instruction set and a high quality compiler should outperform a computer with a complex instruction set built at a comparable cost. The researchers at Berkeley supported their arguments with experimental results from an architectural simulator and from a prototype microprocessor built with a reduced instruction set. Commercial RISC processors have been produced by several manufacturers, including Hewlett Packard, Motorla, and IBM.

In a sense, the RISC approach reintroduces a very old idea: that software should be used to hide hardware complexities. This is the argument used by von Neumann to favor binary computers, despite the difficulty of converting between binary and decimal (Burks, Goldstine, and von Neumann 1946). Von Neumann had argued that it would be more efficient to build binary machines and rely on software to convert data from decimal to binary and back. A major argument favoring RISC is that RISC programmers can rely on compiler software to hide the complexity of using such a simple instruction set.

Historically, instruction sets have grown complex in an attempt to simplify programming. Today, few programmers deal directly with an instruction set. Instead, most programmers use high level languages that are converted to machine language by compiler software. A major argument favoring complex instruction sets today is that a common computation will usually be performed quicker by a single built-in instruction than by a sequence of instructions that must be executed individually. But the effectiveness of such instructions depends on how well the compiler software can exploit them, making this issue transcend the old boundaries of computer architecture.

Perspective

Over the past decades, the cost of computing hardware has dropped dramatically and consistently. It is widely believed that this trend will continue at least until the year 2000. Much of the cost improvement in modern circuitry can be traced to continually reduced sizes. At some point, however, fundamental limitations will prevent further improvements. The size of circuit features will eventually be limited by the physics of materials: for example, how do you produce a physical feature that is only a few atoms wide? Similar limitations will be reached on circuit speed; information cannot propagate through a circuit faster than the speed of light. Although switching speed may be the major limitation on computation speed at the moment, speed of light propagation will always place a fundamental limit on circuit speed.

Physical limitations are not the only ones affecting computer architecture. There are issues in engineering economy that push computer architecture in particular directions. For example, economies of scale encourage manufacturers of large scale integrated circuits to concentrate on developing central processors and memories. Integrated circuit design involves extensive engineering design and development; manufacturers cannot afford to develop a large scale integrated circuit unless the circuit has a very large potential market. Memories and central processors are of such a general nature that development expenses are easiest to justify economically. Thus, these are the major building blocks of computer designers who do not build their own integrated circuits.

This concentration on microprocessors and memories has led to increasing interest in multiprocessor systems. Given microprocessors as

building blocks, multiprocessing provides a path towards higher performance computing without the need to design integrated circuits. Improved performance is gained by splitting a computational task among the separate processors in the multiprocessing system. Ideally, the system could always improve its performance by installing additional processors.

Multiprocessor systems, however, are limited by the nature of today's computer applications. Although multiprocessors have been sold commercially for decades, few systems have achieved exceptionally high performance through multiprocessing. The restriction often has been that most programs are designed to operate sequentially and thus cannot take advantage of a multiprocessor. To exploit a multiprocessor, the programmer must write the program so that separate parts of it can be executed simultaneously on separate processors; the more parts that execute simultaneously the faster the program can execute. A basic problem is that few programmers have been trained to program multiprocessors effectively. The few multiprocessor systems that have been commercially successful rely mainly on optimizing compilers and multiprogramming software that improve the performance of conventionally written sequential programs.

In the past, economy has improved with reduced circuit component size while performance has improved with increased parallelism. Parallel arithmetic computers replaced the serial processors in the 1950s since parallel machines were so much faster. Interrupt systems and input/output channels were simply architectural ingenuities to parallelize the processing of input/output operations and computations. The supercomputer architectures with multiple functional units and pipelining are examples of parallelism carried to the execution of one instruction on an entire sequence of data points. The sequential nature of typical computer programs is rapidly becoming the fundamental limitation on increased parallelism and thus on improved performance. Just as the physical limitations are catching up with circuit size and speed improvements, the limitations of sequential programming techniques are being felt in the quest for higher performance.

Programming traditions and standards play an ever increasing role in computer architecture, primarily by limiting the scope of viable options. Architects and designers can produce multiprocessors to solve specific problems at arbitrarily high speeds if cost is no object. The challenge, however, is to produce computing systems that yield high performance when solving a variety of problems. Programmers know how to use sequential, high level languages such as FORTRAN, making them a crucial part of modern computing systems. Widely available systems that use Unix® or MS-DOS® software also have an effect: computer design today is often limited to techniques that can exploit existing software and skills.

Acknowledgments

This work was supported by the Charles Babbage Institute of the University of Minnesota, Minneapolis, Minnesota, 55455. The work is part of the National Collecting Strategy, a three year program to develop a national collecting strategy for preserving the historic records of computing. This work is made possible through the generous support of the AT&T Foundation, IBM Corporation, the Andrew W. Mellon Foundation, and Unisys Corporation.

The author would also like to express his gratitude to the Directors and staff of the Charles Babbage Institute for their help in this research. Dr. William Aspray, the Associate Director, provided guidance and encouragement that was crucial to this project. Bruce Bruemmer, the Archivist, also provided valued assistance.

The author also appreciates the contributions of his employer, the Advanced Systems Center of FMC Corporation, in the production of the final draft.

Unix is a trademark of AT&T Bell Laboratories. MS-DOS is a trademark of Microsoft Corporation. Apple II, Lisa, and Macintosh are trademarks of Apple Computer.

References

NOTE: *Many of these references have been republished in readily available books about computer architecture. The two books, entitled* Computer Structures, *are particularly valuable in this regard; the following abbreviations are used in a bibliographic entry if the paper also appears in these volumes.*

B&N Bell and Newell, 1971
SB&N Siewiorek, Bell, and Newell, 1982

ACM. 1986. *Proc. ACM Conference on the History of Personal Workstations,* Association for Computing Machinery, New York.

Amdahl, G. M., G. A. Blaauw, and F. P. Brooks, Jr.

1964. "Architecture of System/360," *IBM Journal of Research and Development*, vol. 8, no. 2, April, pp. 87–101.

Allen, F. E. 1984. "A Technological Review of the Early FORTRAN Compilers," *Annals of the History of Computing*, vol. 6, no. 1, January, pp. 22–26.

Apollo. 1981. "Apollo System User's Guide," Apollo Computer, Chelmsford, Massachusetts.

Backus, J. 1979. "The History of FORTRAN I, II, and III," *Annals of the History of Computing*, vol. 1, no. 1, July, pp. 21–37.

Baecker, R. and W. A. S. Buxton, 1987. "Case Study D: The Star, the Lisa, and the Macintosh," in *Readings in Human-Computer Interaction*, Baecker and Buxton, eds., Morgan Kaufmann Publishers, Los Altos, California.

Barnes, G. H., R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, 1968. "The Illiac IV Computer," *IEEE Transactions C-17*, vol. 8, August, pp. 746–767; also in B&N, pp. 320–333.

Bashe, C. J., L. R. Johnson, J. H. Palmer, and E. W. Pugh. 1986. *IBM's Early Computers*, MIT Press, Cambridge, Massachusetts.

Bell, C. G., R. Cady, H. McFarland, B. A. Delagi, J. F. O'Loughlin, R. Noonan and W. A. Wulf. 1970. "A New Architecture for Minicomputers—The DEC PDP-11," Proc. SJCC, 1970, pp. 657–675; reprinted in *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, Massachusetts, 1978; also in SB&N, pp. 649–661.

Bell, C. G., and A. Newell. 1971. *Computer Structures: Readings and Examples*, McGraw-Hill, New York.

Bell, C. G., G. Butler, R. Gray, J. E. McNamara, D. Vonada, and R. Wilson. 1978. "The PDP-1 and Other 18-Bit Computers," in *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, Massachusetts.

Bell, C. G., J. C. Mudge, and J. E. McNamara. 1978. *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, Massachusetts.

Bell, C. G. and J. E. McNamara. 1978. "The PDP-8 and Other 12-Bit Computers," in *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, Massachusetts.

Bouknight, W. J., S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick. 1972. "The Illiac IV System," *Proc. IEEE*, April, pp. 369–379; excerpt in SB&N, pp. 306–316.

Blaauw, G. A., and F. P. Brooks, Jr. 1964. "The structure of System/360," *IBM Systems Journal*, vol. 3, no. 2, pp. 119–135; also in B&N, pp. 588–601.

Blachman, N. M. 1953. "A Survey of Automatic Digital Computers," Publication PB 111293, Office of Technical Services, Department of Commerce, Washington, D.C.

Bloch, E. 1959. "The engineering design of the Stretch computer," *Proc. FJCC*, pp. 48–59; also in B&N, pp. 421–439.

Bowden, B. V. (ed.). 1953. *Faster than Thought*, Pitman Publishing Corporation, New York.

Buchholz, W. (ed.). 1962. *Planning a Computer System: Project Stretch*, McGraw-Hill, New York, 1962.

Burks, A. W., H. H. Goldstine, and J. von Neumann. 1946. "Preliminary discussion of the logical design of an electronic computing instrument," Institute for Advanced Study, Princeton, New Jersey; also in B&N, pp. 92–119.

Campbell-Kelly, M. 1980. "Programming the Mark I: Early Programming Activity at the University of Manchester," *Annals of the History of Computing*, vol. 2, no. 2, April, pp. 130–168.

CDC. 1968. "Control Data 6400/6500/6600 Computer Systems Reference Manual," Pub. No. 60100000, Revision F, Control Data Corporation, St. Paul, Minnesota.

Clark, W. A. 1957. "The Lincoln TX-2 Computer Development," *Proc. WJCC*, pp. 143–145.

Dean, L. C. 1973. "Texas Instruments Advanced Scientific Computuer," *Informatie Jrg.*, vol. 15, no. 4, April, pp. 191–193; excerpted in SB&N, pp. 759–762.

Dunwell, S. W. 1956. "Design Objectives of the IBM Stretch Computer," *Proc. EJCC*, pp. 20–22.

Eckert, J. P. Jr., J. R. Weiner, H. F. Welsh, and H. F. Mitchell. 1951. "The Univac System," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 6–16; also in B&N, pp. 157–169.

Eckert, J. P. Jr., J. C. Chu, A. B. Tonik, and W. J. Schmidt. 1959. "Design of Univac-LARC System, Part I," *Proc. EJCC*, pp. 59–65.

Eckert, J. P. Jr. 1956. "Univac-LARC: the Next Step in Computer Design," *Proc. EJCC*, pp. 16–19.

Ein-dor, P. 1985. "Grosch's Law Revisited," *Comm. ACM*, Vol. 28, No. 2, Feb., pp. 142–151.

Engineering Research Associates. 1950. *High Speed Computing Devices*, McGraw-Hill, New York; republished by Tomash Publishers, 1983.

Everett, R. R. and F. E. Swain. 1947. "Whirlwind I Computer Block Diagrams," Report R-127, two volumes, Servomechanisms Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, September, [report to the Office of Naval Research, classified CONFIDENTIAL and declassified by 1960].

Everett, R. R. 1951. "The Whirlwind I Computer," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 70–74; also in B&N, pp. 137–145.

Everett, R. R. (ed.). 1983. "Special Issue: SAGE (Semi-Automatic Ground Environment)," *Annals of the History of Computing*, vol. 5, no. 4, October.

Forgie, J. W. 1957. "The Lincoln TX-2 Input/Output System," *Proc. WJCC*.

Goldstine, H. H. and A. Goldstine. 1946. "The Electronic Numerical Integrator and Computer (ENIAC)," *Mathematical Tables and Aids to Computation*, vol. 2, July, pp. 97–110.

Goldstine, H. H. 1972. *The Computer from Pascal to von Neumann*, Princeton University Press, New Jersey.

Gruenberger, F. J. 1979. "The History of the JOHNNIAC," *Annals of the History of Computing*, vol. 1, no. 1, July, pp. 49–64.

Haanstra, J. W. et al. 1961. "Processor Products—Final Report of the SPREAD Task Group," IBM Corporation, December, reprinted in *Annals of the History of Computing*, vol. 5, no. 1, January 1983.

Harvard Computation Laboratory. 1944 *A Manual of Operation for the Automatic Sequence Controlled Calculator*, Harvard University, Cambridge, Massachusetts, reprinted by MIT Press, Cambridge, Massachusetts, 1985.

Harvard Computation Laboratory. 1947. *Proceedings of a Symposium on Large-Scale Digital Calculating Machinery*, Harvard University, Cambridge, Massachusetts; reprinted by MIT Press, Cambridge, Massachusetts, 1985.

Howarth, D. J., R. B. Payne, and F. H. Sumner. 1961. "The Manchester University Atlas Operating System, Part II: User's Description," *Comp. J.*, vol. 4, October, pp. 226–229.

Hurd, C. C. 1980. "Computer Development at IBM," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and Gian-Carlo Rota, eds., Academic Press, New York.

Hurd, C. C. 1981. "Early IBM Computers: Edited Testimony," *Annals of the History of Computing*, vol. 3, no. 2, April, pp. 163–182.

Hurd, C. C. (ed.). 1986. "Special Issue: IBM 650," *Annals of the History of Computing*, vol. 8, no. 1, January.

IBM. 1970. "IBM System/360 Principles of Operation," File No. S360-01, Ninth Edition, IBM Corporation, Poughkeepsie, New York, November.

IBM. 1974. "IBM System/370 Principles of Operation," File No. S/370-01, Fourth Edition, IBM Corporation, Poughkeepsie, New York, November.

IBM. 1981. "Product Announcement: The IBM Personal Computer," International Business Machines Corporation, Data Processing Division, White Plains, New York, 12 August.

Kilburn, T., D. J. Howarth, R. B. Payne, and F. H. Sumner. 1961. "The Manchester University Atlas Operating System, Part I: Internal Organization," *Comp. J.*, vol. 4, October, pp. 222–225.

Kilburn, T., D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner. 1962. "One-level storage system," *IRE Transactions EC-11*, vol. 2, April, pp. 223–235; also in SB&N, pp. 135–148.

Knight, K. E., and R. P. Cerveny. 1983. "Grosch's Law," in *Encyclopedia of Computer Science and Engineering*, second edition, A. Ralston and E. D. Reilly, Jr., (eds.), Van Nostrand Reinhold, New York.

Lavington, S. 1975. "A History of Manchester Computers," NCC Publications; excerpted in SB&N, pp. 107–108.

Lee, J. A. N. 1985. "Reviews: *Hackers* by Steven Levy," *Annals of the History of Computing*, vol. 7, no. 3, July, pp. 270–272.

Leonard, T. E. (ed.). 1987. *VAX Architecture Reference Manual*, Digital Press, Bedford, Massachusetts.

Levy, S. 1984. *Hackers, Heroes of the Computer Revolution*, Dell Publishing Co., New York.

Lonergan, W. and P. King. 1961. "Design of the B5000 System," *Datamation*, vol. 7, no. 5, May, pp. 28–32; also in B&N, pp. 267–273; and in SB&N, pp. 129–134.

Mersel, J. 1956. "Program Interrupt on the Univac Scientific Computer," *Proc. WJCC*, San Francisco, California, February, pp. 52–53.

Mitchell, J. L. and K. H. Olsen. 1956. "TX-0: A Transistor Computer," *Proc. EJCC*, pp. 93–101.

Moore School of Electrical Engineering. 1946. "Theory and Techniques for Design of Electronic Digital Computers," lecture notes from a special course, 4 volumes, University of Pennsylvania, Philadelphia, 8 July to 31 August; reprinted as *The Moore School Lectures*, M. Campbell-Kelly and M. R. Williams, eds., MIT Press, Cambridge, Massachusetts.

Morse, S. P., W. B. Pohlman, and B. W. Ravenel. 1978. "The Intel 8086 Microprocessor: A 16-Bit Evolution of the 8080," *IEEE Computer*, June, pp. 18–27.

Mullaney, F. C., 1951. "Design Features of the ERA 1101 Computer," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 43–49.

Myer, T. H. and I. E. Sutherland. 1968. "On the Design of Display Processors," *Comm. ACM*, vol. 11, no. 6, June, pp. 410–414.

Patterson, D. A. and C. H. Séquin. 1982. "A VLSI RISC," *IEEE Computer*, vol. 15, no. 9, September, pp. 8–21.

Poorte, G. E. 1951. "The Operation and Logic of the MARK III Electronic Calculator in View of Operating Experience," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 50–55.

Randell, B. 1983. "Digital Computers, History," in *Encyclopedia of Computer Science and Engineering*, second edition, Anthony Ralston and Edwin D. Reilly, Jr. (eds.), Van Nostrand Reinhold, New York, pp. 532–535.

Randell, B. and L. J. Russell. 1964. *ALGOL 60 Implementation*, Academic Press, London.

Redmond, K. C. and T. M. Smith. 1980. *Project Whirlwind: The History of a Pioneer Computer*, Digital Press, Bedford, Massachusetts.

Rees, M. 1982. "The Computing Program of ONR, 1946–1953," *Annals of the History of Computing*, vol. 4, no. 2, April, pp. 102–120.

Robertson, J. E. 1980. "The ORDVAC and the ILLIAC," in *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G. Rota (eds.), Academic Press, New York, pp. 347–364.

Rosen, S. 1969. "Electronic Computers: A Historical Survey," *Computing Surveys*, vol. 1, no. 1, March, pp. 7–36.

Rosin, R. F. (ed.). 1987. "Special Issue: The Burroughs B5000," *Annals of the History of Computing*, vol. 9, no. 1.

Russell, R. M. 1978. "The CRAY-1 Computer System,"

Comm. ACM, vol. 21, no. 1, January 1978, pp. 63–72; also in SB&N, pp. 743–752.

Sammet, J. E. 1987. "Answers to Self-Study Questions," Vol. 9, No. 2, *Ann. Hist. Comp.*, p. 217.

Sharpless, T. K. et al. 1946. "Progress Report 2 on The EDVAC (Electronic Discrete Variable Computer)," 2 volumes, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, 30 June, [report to the Army Ordinance Department, classified CONFIDENTIAL and declassified by 1960].

Siewiorek, D. P., C. G. Bell, and A. Newell. 1982. *Computer Structures: Principles and Examples*, McGraw-Hill, New York.

Smith, A. E. 1948. "A Survey of Large Scale Computers and Computer Projects," Office of Naval Research, Navy Department, Washington, D.C., August [classified RESTRICTED when published and declassified in 1955].

Stern, N. 1981. *From ENIAC to UNIVAC, An Appraisal of the Eckert-Mauchly Computers*, Digital Press, Bedford, Massachusetts.

Sumner, F. H., G. Haley, and E. C. Y. Chen. 1962. "The Central Control Unit of the 'Atlas' Computer," *Proc. IFIP Cong.*, pp. 657–662.

Texas Instruments, Inc. 1975. *TMS1000 Programmer's Reference Manual;* excerpted in SB&N, pp. 587–599.

Thacker, C. P., E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs. 1979. "Alto: A Personal Computer," Xerox Corporation, Palo Alto, California; also in SB&N, pp. 549–572.

Thornton, J. E. 1964. "Parallel Operation in the Control Data 6600," *Proc. FJCC*, vol. 26, pt. 2, pp. 33–40; also in SB&N, pp. 730–736.

Thornton, J. E. 1980. "The CDC 6600 Project," *Annals of the History of Computing*, vol. 2, no. 4, October, pp. 338–348.

Tomash, E. and A. A. Cohen. 1979. "The Birth of an ERA: Engineering Research Associates, Inc. 1946–1955," *Annals of the History of Computing*, vol. 1, no. 2, October, pp. 83–97.

von Neumann, J. 1945. "First Draft of a Report on the EDVAC," 30 June; reprinted in Stern, *From ENIAC to UNIVAC: A Case Study in the History of Technology*, Digital Press, Bedford, Massachusetts, 1981.

Warren, J. 1977. "Personal and hobby computing: an overview," *IEEE Computer*, vol. 10, no. 3, March, pp. 10–22.

Watson, W. J. 1972. "The TI ASC: A Highly Modular and Flexible Super Computer Architecture," *Proc. AFIPS FJCC*, pp. 221–228; also in SB&N, pp. 753–759.

Weik, M. H. 1961. "A third survey of domestic electronic digital computing systems," Report 1114, Ballistic Research Lab, Aberdeen Proving Grounds, Aberdeen, Maryland, March.

Wexelblat, R. L. (ed.) 1981. *Proc. History of Programming Languages Conference*, Academic Press, New York; preprint appeared in *ACM SIGPLAN Notices*, vol. 13, no. 8, August 1978.

Wilkes, M. V. 1951. "The Best Way to Design An Automatic Calculating Machine," *Manchester University Computer Inaugural Conference*, Ferranti Ltd., London, July; also in *Annals of the History of Computing*, vol. 8, no. 2, April, 1986, pp. 118–121.

Wilkes, M. V. 1951. "The EDSAC Computer," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 79–83.

Wilkes, M. V., D. J. Wheeler, and S. Gill. 1951. *The Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley, Cambridge.

Wilkes, M. V. and J. B. Stringer. 1953. "Micro-Programming and the Design of the Control Circuits in an Electronic Digital Computer," *Proceedings of the Cambridge Philosophical Society*, pt. 2, vol. 49, April, pp. 230–238; reprinted in *Annals of the History of Computing*, vol. 8, no. 2, April, 1986, pp. 121–126; also in B&N, pp. 335–340; and in SB&N, pp. 158–163.

Williams, F. C. and T. Kilburn, 1951. "The University of Manchester Computing Machine," *Proc. Joint AIEE-IRE Computer Conference*, Philadelphia, December, pp. 57–61.

Wozniak, S. 1977. "The Apple II," *Byte*, vol. 2, no. 5, May.